EXHIBIT 1

Exhibit 1: Comparison of Asserted Deque Help Code, Accused BrowserStack Help Text Code, and WCAG Standards

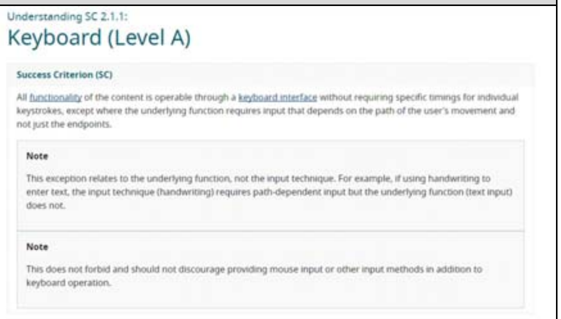
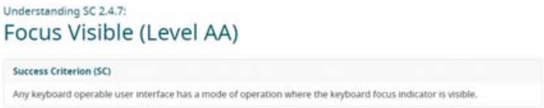
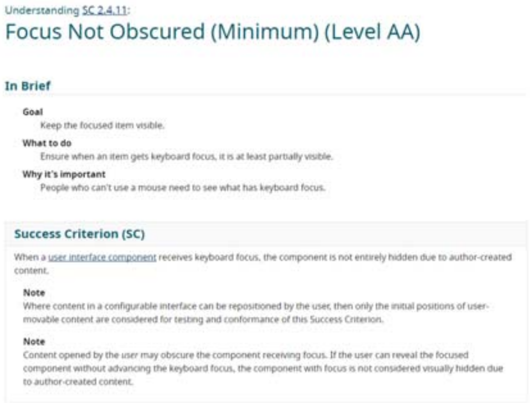
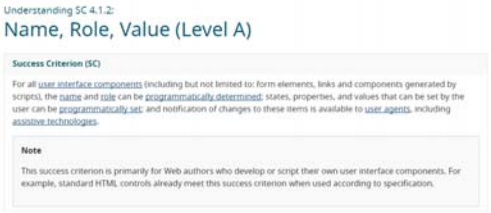
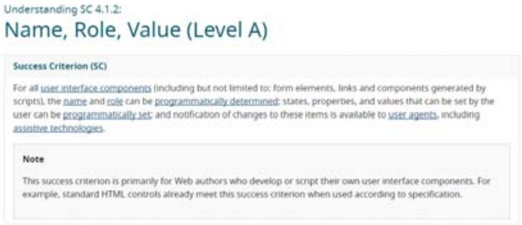
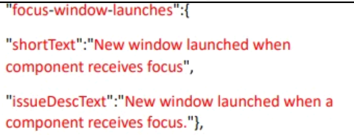
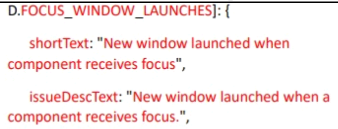
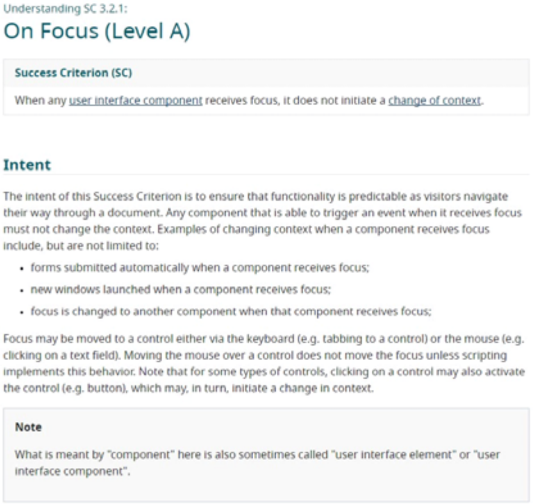
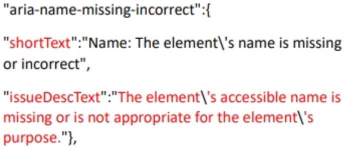
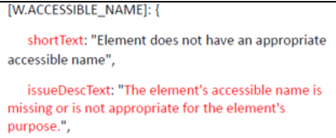
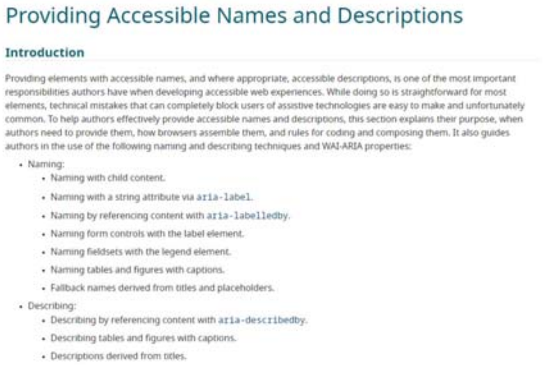
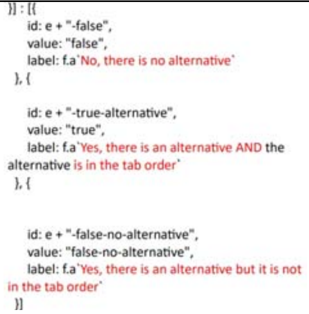
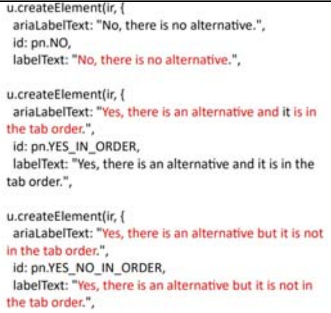
Code Segment	Asserted Deque Help Text Code	Accused BrowserStack Help Text Code	WCAG Standard
Deque 4.9.0 "vendor.bundle.js" // BroswerStack 1.0.0.4: "IGT Rules.js"			
1	"keyboard-inaccessible":{ "shortText":"Action cannot be performed by keyboard alone", "issueDescText":"General - There is no way to perform the function using only the keyboard."},	[D.KEYBOARD_INACCESSIBLE]: { shortText: "Function cannot be performed by keyboard alone", issueDescText: "There is no way to perform the function using only the keyboard on the same screen or on a qualifying conforming alternate version.",	 Understanding SC 2.1.1: Keyboard (Level A) Success Criterion (SC) All functionality of the content is operable through a keyboard interface without requiring specific timings for individual keystrokes, except where the underlying function requires input that depends on the path of the user's movement and not just the endpoints. Note This exception relates to the underlying function, not the input technique. For example, if using handwriting to enter text, the input technique (handwriting) requires path-dependent input but the underlying function (text input) does not. Note This does not forbid and should not discourage providing mouse input or other input methods in addition to keyboard operation.
2	"focus-indicator-missing":{ "shortText":"Focus indicator is missing", "issueDescText":"The visual focus indicator is missing altogether."},	[D.FOCUS_INDICATOR_MISSING]: { shortText: "Focus indicator is missing", issueDescText: "The visual focus indicator is missing altogether.",	 Understanding SC 2.4.7: Focus Visible (Level AA) Success Criterion (SC) Any keyboard operable user interface has a mode of operation where the keyboard focus indicator is visible.
3	"focus-on-hidden-item":{ "shortText":"Hidden or empty element receives focus", "issueDescText":"Keyboard focus or touch screen swiping falls on an element that is hidden or empty."},	[D.FOCUS_ON_HIDDEN_ITEM]: { shortText: "Hidden or empty element receives focus", issueDescText: "Keyboard focus or touch screen swiping falls on an element that is hidden or empty."	 Understanding SC 2.4.11: Focus Not Obscured (Minimum) (Level AA) In Brief Goal Keep the focused item visible. What to do Ensure when an item gets keyboard focus, it is at least partially visible. Why it's important People who can't use a mouse need to see what has keyboard focus. Success Criterion (SC) When a user interface component receives keyboard focus, the component is not entirely hidden due to author-created content. Note Where content in a configurable interface can be repositioned by the user, then only the initial positions of user-movable content are considered for testing and conformance of this Success Criterion. Note Content opened by the user may obscure the component receiving focus. If the user can reveal the focused component without advancing the keyboard focus, the component with focus is not considered visually hidden due to author-created content.
4	"aria-role-missing":{ "shortText":"Role: The element\'s role is missing or incorrect", "issueDescText":"The element\'s role is missing or is not appropriate for the element\'s function."},	[D.ARIA_ROLE_MISSING]: { shortText: "Role: The element's role is missing or incorrect", issueDescText: "The element's role is missing or is not appropriate for the element's function.",	 Understanding SC 4.1.2: Name, Role, Value (Level A) Success Criterion (SC) For all user interface components (including but not limited to: form elements, links and components generated by scripts), the name and role can be programmatically determined; states, properties, and values that can be set by the user can be programmatically set; and notification of changes to these items is available to user agents, including assistive technologies. Note This success criterion is primarily for Web authors who develop or script their own user interface components. For example, standard HTML controls already meet this success criterion when used according to specification.
5	"aria-state-property-missing":{ "shortText":"States/Properties: The element has missing or incorrect states or properties", "issueDescText":"The element has missing or incorrect states or properties that are necessary for screen reader users to interact with or understand the content conveyed by the element."},	[W.ACCESSIBLE_STATES]: { shortText: "Element has missing or incorrect states", issueDescText: "The element has missing or incorrect states or properties that are necessary for screen reader users to interact with or understand the content conveyed by the element."	 Understanding SC 4.1.2: Name, Role, Value (Level A) Success Criterion (SC) For all user interface components (including but not limited to: form elements, links and components generated by scripts), the name and role can be programmatically determined; states, properties, and values that can be set by the user can be programmatically set; and notification of changes to these items is available to user agents, including assistive technologies. Note This success criterion is primarily for Web authors who develop or script their own user interface components. For example, standard HTML controls already meet this success criterion when used according to specification.

Exhibit 1: Comparison of Asserted Deque Help Code, Accused BrowserStack Help Text Code, and WCAG Standards

Code Segment	Asserted Deque Help Text Code	Accused BrowserStack Help Text Code	WCAG Standard
6	 "focus-window-launches":{ "shortText":"New window launched when component receives focus", "issueDescText":"New window launched when a component receives focus."},	 D.FOCUS_WINDOW_LAUNCHES]: { shortText: "New window launched when component receives focus", issueDescText: "New window launched when a component receives focus.",	 Understanding SC 3.2.1: On Focus (Level A) Success Criterion (SC) When any user interface component receives focus, it does not initiate a change of context. Intent The intent of this Success Criterion is to ensure that functionality is predictable as visitors navigate their way through a document. Any component that is able to trigger an event when it receives focus must not change the context. Examples of changing context when a component receives focus include, but are not limited to: • forms submitted automatically when a component receives focus; • new windows launched when a component receives focus; • focus is changed to another component when that component receives focus; Focus may be moved to a control either via the keyboard (e.g. tabbing to a control) or the mouse (e.g. clicking on a text field). Moving the mouse over a control does not move the focus unless scripting implements this behavior. Note that for some types of controls, clicking on a control may also activate the control (e.g. button), which may, in turn, initiate a change in context. Note What is meant by "component" here is also sometimes called "user interface element" or "user interface component".
7	 "aria-name-missing-incorrect":{ "shortText":"Name: The element\'s name is missing or incorrect", "issueDescText":"The element\'s accessible name is missing or is not appropriate for the element\'s purpose."},	 [W.ACCESSIBLE_NAME]: { shortText: "Element does not have an appropriate accessible name", issueDescText: "The element's accessible name is missing or is not appropriate for the element's purpose.",	 Providing Accessible Names and Descriptions Introduction Providing elements with accessible names, and where appropriate, accessible descriptions, is one of the most important responsibilities authors have when developing accessible web experiences. While doing so is straightforward for most elements, technical mistakes that can completely block users of assistive technologies are easy to make and unfortunately common. To help authors effectively provide accessible names and descriptions, this section explains their purpose, when authors need to provide them, how browsers assemble them, and rules for coding and composing them. It also guides authors in the use of the following naming and describing techniques and WAI-ARIA properties: • Naming: • Naming with child content. • Naming with a string attribute via aria-label. • Naming by referencing content with aria-labelledby. • Naming form controls with the label element. • Naming fieldsets with the legend element. • Naming tables and figures with captions. • Fallback names derived from titles and placeholders. • Describing: • Describing by referencing content with aria-describedby. • Describing tables and figures with captions. • Descriptions derived from titles.
Deque 4.9.0 "panel.bundle.js" // BroswerStack 1.0.0.4: "default.js"			

8	
	}] : [{
	id: e + "-false",
	value: "false",
	label: f.a`No, there is no alternative`
	}, {
	id: e + "-true-alternative",
	value: "true",
	label: f.a`Yes, there is an alternative AND the alternative is in the tab order`
	}, {
	id: e + "-false-no-alternative",
	value: "false-no-alternative",
	label: f.a`Yes, there is an alternative but it is not in the tab order`
	}]
		
		u.createElement(ir, {
		ariaLabelText: "No, there is no alternative.",
		id: pn.NO,
		labelText: "No, there is no alternative.",
		u.createElement(ir, {
		ariaLabelText: "Yes, there is an alternative and it is in the tab order.",
		id: pn.YES_IN_ORDER,
		labelText: "Yes, there is an alternative and it is in the tab order.",
		u.createElement(ir, {
		ariaLabelText: "Yes, there is an alternative but it is not in the tab order.",
		id: pn.YES_NO_IN_ORDER,
		labelText: "Yes, there is an alternative but it is not in the tab order.",	

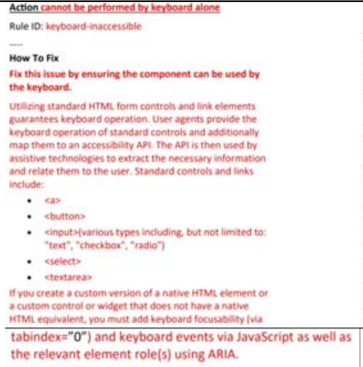
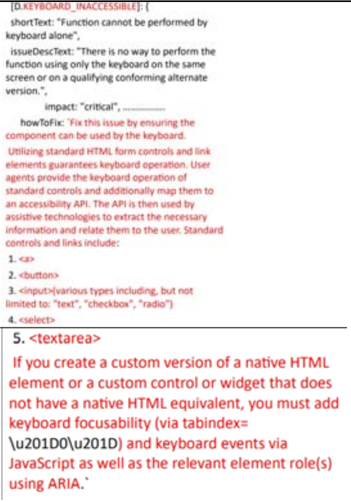
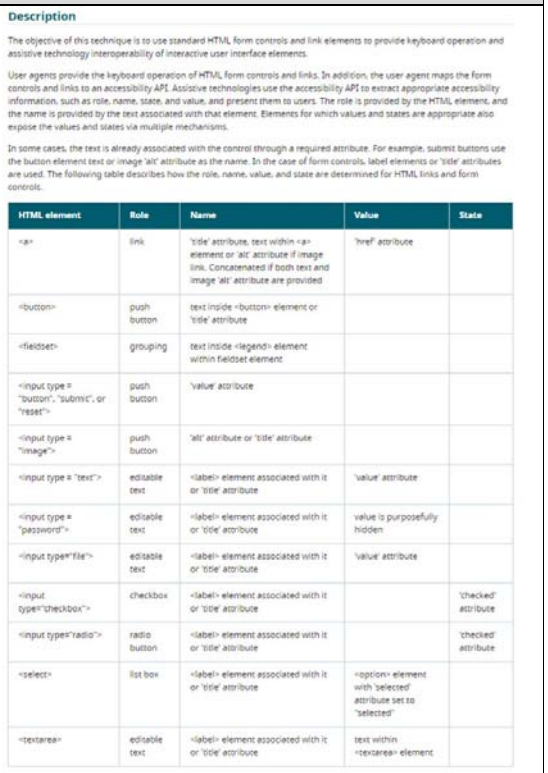
<table>
<tr><th colspan="4">Exhibit 1: Comparison of Asserted Deque Help Code, Accused BrowserStack Help Text Code, and WCAG Standards</th></tr>
<tr><th>Code Segment</th><th>Asserted Deque Help Text Code</th><th>Accused BrowserStack Help Text Code</th><th>WCAG Standard</th></tr>
<tr><td colspan="4">Deque Rules Help Pages // BroswerStack 1.0.0.4: "IGTRules.js"</td></tr>
<tr>
<td>9</td>
<td>
Action cannot be performed by keyboard alone
Rule ID: keyboard-inaccessible
.....
How To Fix
Fix this issue by ensuring the component can be used by the keyboard.
Utilizing standard HTML form controls and link elements guarantees keyboard operation. User agents provide the keyboard operation of standard controls and additionally map them to an accessibility API. The API is then used by assistive technologies to extract the necessary information and relate them to the user. Standard controls and links include:
• &lt;a&gt;
• &lt;button&gt;
• &lt;input&gt;(various types including, but not limited to: "text", "checkbox", "radio")
• &lt;select&gt;
• &lt;textarea&gt;
If you create a custom version of a native HTML element or a custom control or widget that does not have a native HTML equivalent, you must add keyboard focusability (via tabindex="0") and keyboard events via JavaScript as well as the relevant element role(s) using ARIA.</td>
<td>
[D.KEYBOARD_INACCESSIBLE]: {
shortText: "Function cannot be performed by keyboard alone",
issueDescText: "There is no way to perform the function using only the keyboard on the same screen or on a qualifying conforming alternate version.",
impact: "critical", ...
howToFix: `Fix this issue by ensuring the component can be used by the keyboard.
Utilizing standard HTML form controls and link elements guarantees keyboard operation. User agents provide the keyboard operation of standard controls and additionally map them to an accessibility API. The API is then used by assistive technologies to extract the necessary information and relate them to the user. Standard controls and links include:
1. &lt;a&gt;
2. &lt;button&gt;
3. &lt;input&gt;(various types including, but not limited to: "text", "checkbox", "radio")
4. &lt;select&gt;
5. &lt;textarea&gt;
If you create a custom version of a native HTML element or a custom control or widget that does not have a native HTML equivalent, you must add keyboard focusability (via tabindex= \u201D0\u201D) and keyboard events via JavaScript as well as the relevant element role(s) using ARIA.`</td>
<td>
Description
The objective of this technique is to use standard HTML form controls and link elements to provide keyboard operation and assistive technology interoperability of interactive user interface elements.
User agents provide the keyboard operation of HTML form controls and links. In addition, the user agent maps the form controls and links to an accessibility API. Assistive technologies use the accessibility API to extract appropriate accessibility information, such as role, name, state, and value, and present them to users. The role is provided by the HTML element, and the name is provided by the text associated with that element. Elements for which values and states are appropriate also expose the values and states via multiple mechanisms.
In some cases, the text is already associated with the control through a required attribute. For example, submit buttons use the button element text or image 'alt' attribute as the name. In the case of form controls, label elements or 'title' attributes are used. The following table describes how the role, name, value, and state are determined for HTML links and form controls.
<table>
<tr><th>HTML element</th><th>Role</th><th>Name</th><th>Value</th><th>State</th></tr>
<tr><td>&lt;a&gt;</td><td>link</td><td>'title' attribute, text within &lt;a&gt; element or 'alt' attribute if image link. Concatenated if both text and image 'alt' attribute are provided</td><td>'href' attribute</td><td></td></tr>
<tr><td>&lt;button&gt;</td><td>push button</td><td>text inside &lt;button&gt; element or 'title' attribute</td><td></td><td></td></tr>
<tr><td>&lt;fieldset&gt;</td><td>grouping</td><td>text inside &lt;legend&gt; element within fieldset element</td><td></td><td></td></tr>
<tr><td>&lt;input type = "button", "submit", or "reset"&gt;</td><td>push button</td><td>'value' attribute</td><td></td><td></td></tr>
<tr><td>&lt;input type = "image"&gt;</td><td>push button</td><td>'alt' attribute or 'title' attribute</td><td></td><td></td></tr>
<tr><td>&lt;input type = "text"&gt;</td><td>editable text</td><td>&lt;label&gt; element associated with it or 'title' attribute</td><td>'value' attribute</td><td></td></tr>
<tr><td>&lt;input type = "password"&gt;</td><td>editable text</td><td>&lt;label&gt; element associated with it or 'title' attribute</td><td>value is purposefully hidden</td><td></td></tr>
<tr><td>&lt;input type="file"&gt;</td><td>editable text</td><td>&lt;label&gt; element associated with it or 'title' attribute</td><td>'value' attribute</td><td></td></tr>
<tr><td>&lt;input type="checkbox"&gt;</td><td>checkbox</td><td>&lt;label&gt; element associated with it or 'title' attribute</td><td></td><td>'checked' attribute</td></tr>
<tr><td>&lt;input type="radio"&gt;</td><td>radio button</td><td>&lt;label&gt; element associated with it or 'title' attribute</td><td></td><td>'checked' attribute</td></tr>
<tr><td>&lt;select&gt;</td><td>list box</td><td>&lt;label&gt; element associated with it or 'title' attribute</td><td>&lt;option&gt; element with 'selected' attribute set to "selected"</td><td></td></tr>
<tr><td>&lt;textarea&gt;</td><td>editable text</td><td>&lt;label&gt; element associated with it or 'title' attribute</td><td>text within &lt;textarea&gt; element</td><td></td></tr>
</table></td>
</tr>
</table>

Exhibit 1: Comparison of Asserted Deque Help Code, Accused BrowserStack Help Text Code, and WCAG Standards

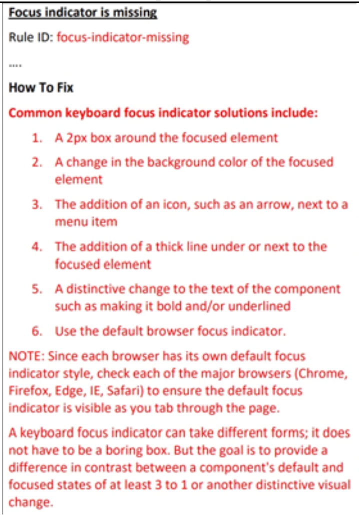
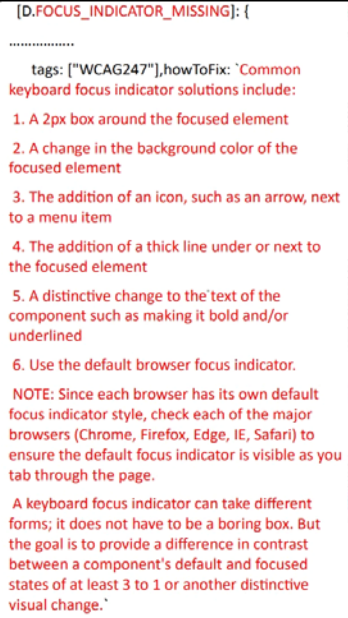
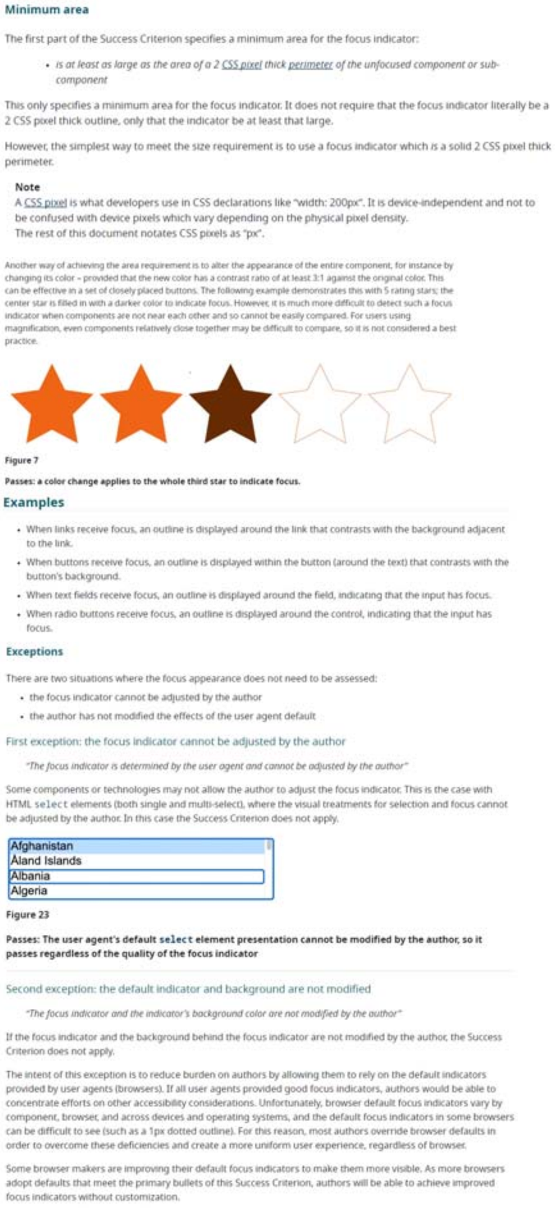
Code Segment	Asserted Deque Help Text Code	Accused BrowserStack Help Text Code	WCAG Standard
10	 Focus indicator is missing Rule ID: focus-indicator-missing …. **How To Fix** **Common keyboard focus indicator solutions include:** 1. A 2px box around the focused element 2. A change in the background color of the focused element 3. The addition of an icon, such as an arrow, next to a menu item 4. The addition of a thick line under or next to the focused element 5. A distinctive change to the text of the component such as making it bold and/or underlined 6. Use the default browser focus indicator. NOTE: Since each browser has its own default focus indicator style, check each of the major browsers (Chrome, Firefox, Edge, IE, Safari) to ensure the default focus indicator is visible as you tab through the page. A keyboard focus indicator can take different forms; it does not have to be a boring box. But the goal is to provide a difference in contrast between a component's default and focused states of at least 3 to 1 or another distinctive visual change.	 [D.FOCUS_INDICATOR_MISSING]: { …………….. tags: ["WCAG247"],howToFix: `Common keyboard focus indicator solutions include: 1. A 2px box around the focused element 2. A change in the background color of the focused element 3. The addition of an icon, such as an arrow, next to a menu item 4. The addition of a thick line under or next to the focused element 5. A distinctive change to the`text of the component such as making it bold and/or underlined 6. Use the default browser focus indicator. NOTE: Since each browser has its own default focus indicator style, check each of the major browsers (Chrome, Firefox, Edge, IE, Safari) to ensure the default focus indicator is visible as you tab through the page. A keyboard focus indicator can take different forms; it does not have to be a boring box. But the goal is to provide a difference in contrast between a component's default and focused states of at least 3 to 1 or another distinctive visual change.`	 **Minimum area** The first part of the Success Criterion specifies a minimum area for the focus indicator: - *is at least as large as the area of a 2 CSS pixel thick perimeter of the unfocused component or sub-component* This only specifies a minimum area for the focus indicator. It does not require that the focus indicator literally be a 2 CSS pixel thick outline, only that the indicator be at least that large. However, the simplest way to meet the size requirement is to use a focus indicator which *is* a solid 2 CSS pixel thick perimeter. **Note** A CSS pixel is what developers use in CSS declarations like "width: 200px". It is device-independent and not to be confused with device pixels which vary depending on the physical pixel density. The rest of this document notates CSS pixels as "px". Another way of achieving the area requirement is to alter the appearance of the entire component, for instance by changing its color – provided that the new color has a contrast ratio of at least 3:1 against the original color. This can be effective in a set of closely placed buttons. The following example demonstrates this with 5 rating stars; the center star is filled in with a darker color to indicate focus. However, it is much more difficult to detect such a focus indicator when components are not near each other and so cannot be easily compared. For users using magnification, even components relatively close together may be difficult to compare, so it is not considered a best practice. Figure 7 **Passes: a color change applies to the whole third star to indicate focus.** **Examples** - When links receive focus, an outline is displayed around the link that contrasts with the background adjacent to the link. - When buttons receive focus, an outline is displayed within the button (around the *text*) that contrasts with the button's background. - When text fields receive focus, an outline is displayed around the field, indicating that the input has focus. - When radio buttons receive focus, an outline is displayed around the control, indicating that the input has focus. **Exceptions** There are two situations where the focus appearance does not need to be assessed: - the focus indicator cannot be adjusted by the author - the author has not modified the effects of the user agent default First exception: the focus indicator cannot be adjusted by the author *"The focus indicator is determined by the user agent and cannot be adjusted by the author"* Some components or technologies may not allow the author to adjust the focus indicator. This is the case with HTML `select` elements (both single and multi-select), where the visual treatments for selection and focus cannot be adjusted by the author. In this case the Success Criterion does not apply. Afghanistan Åland Islands Albania Algeria Figure 23 **Passes: The user agent's default `select` element presentation cannot be modified by the author, so it passes regardless of the quality of the focus indicator** Second exception: the default indicator and background are not modified *"The focus indicator and the indicator's background color are not modified by the author"* If the focus indicator and the background behind the focus indicator are not modified by the author, the Success Criterion does not apply. The intent of this exception is to reduce burden on authors by allowing them to rely on the default indicators provided by user agents (browsers). If all user agents provided good focus indicators, authors would be able to concentrate efforts on other accessibility considerations. Unfortunately, browser default focus indicators vary by component, browser, and across devices and operating systems, and the default focus indicators in some browsers can be difficult to see (such as a 1px dotted outline). For this reason, most authors override browser defaults in order to overcome these deficiencies and create a more uniform user experience, regardless of browser. Some browser makers are improving their default focus indicators to make them more visible. As more browsers adopt defaults that meet the primary bullets of this Success Criterion, authors will be able to achieve improved focus indicators without customization.

Exhibit 1: Comparison of Asserted Deque Help Code, Accused BrowserStack Help Text Code, and WCAG Standards

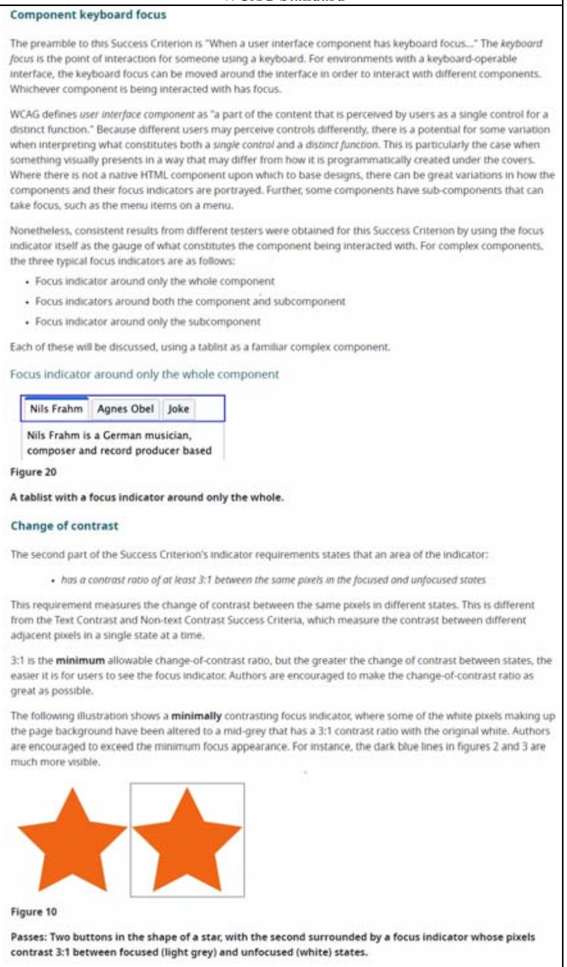
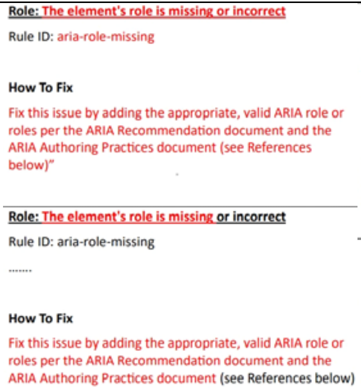
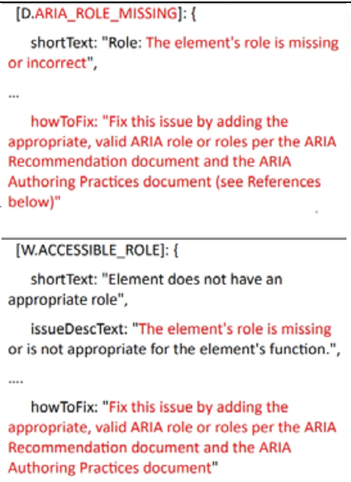
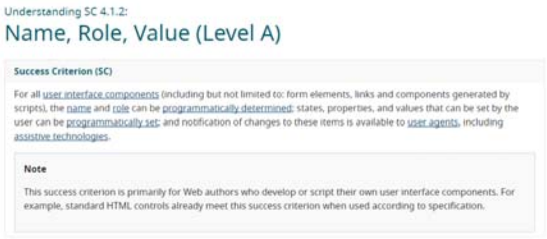
Code Segment	Asserted Deque Help Text Code	Accused BrowserStack Help Text Code	WCAG Standard
			 **Component keyboard focus** The preamble to this Success Criterion is "When a user interface component has keyboard focus..." The *keyboard focus* is the point of interaction for someone using a keyboard. For environments with a keyboard-operable interface, the keyboard focus can be moved around the interface in order to interact with different components. Whichever component is being interacted with has focus. WCAG defines *user interface component* as "a part of the content that is perceived by users as a single control for a distinct function." Because different users may perceive controls differently, there is a potential for some variation when interpreting what constitutes both a *single control* and a *distinct function*. This is particularly the case when something visually presents in a way that may differ from how it is programmatically created under the covers. Where there is not a native HTML component upon which to base designs, there can be great variations in how the components and their focus indicators are portrayed. Further, some components have sub-components that can take focus, such as the menu items on a menu. Nonetheless, consistent results from different testers were obtained for this Success Criterion by using the focus indicator itself as the gauge of what constitutes the component being interacted with. For complex components, the three typical focus indicators are as follows: • Focus indicator around only the whole component • Focus indicators around both the component and subcomponent • Focus indicator around only the subcomponent Each of these will be discussed, using a tablist as a familiar complex component. Focus indicator around only the whole component Nils Frahm Agnes Obel Joke Nils Frahm is a German musician, composer and record producer based **Figure 20** **A tablist with a focus indicator around only the whole.** **Change of contrast** The second part of the Success Criterion's indicator requirements states that an area of the indicator: • *has a contrast ratio of at least 3:1 between the same pixels in the focused and unfocused states* This requirement measures the change of contrast between the same pixels in different states. This is different from the Text Contrast and Non-text Contrast Success Criteria, which measure the contrast between different adjacent pixels in a single state at a time. 3:1 is the **minimum** allowable change-of-contrast ratio, but the greater the change of contrast between states, the easier it is for users to see the focus indicator. Authors are encouraged to make the change-of-contrast ratio as great as possible. The following illustration shows a **minimally** contrasting focus indicator, where some of the white pixels making up the page background have been altered to a mid-grey that has a 3:1 contrast ratio with the original white. Authors are encouraged to exceed the minimum focus appearance. For instance, the dark blue lines in figures 2 and 3 are much more visible. **Figure 10** **Passes: Two buttons in the shape of a star, with the second surrounded by a focus indicator whose pixels contrast 3:1 between focused (light grey) and unfocused (white) states.**

Exhibit 1: Comparison of Asserted Deque Help Code, Accused BrowserStack Help Text Code, and WCAG Standards

Code Segment	Asserted Deque Help Text Code	Accused BrowserStack Help Text Code	WCAG Standard

11	
	Role: The element's role is missing or incorrect
	Rule ID: aria-role-missing
	How To Fix
	Fix this issue by adding the appropriate, valid ARIA role or roles per the ARIA Recommendation document and the ARIA Authoring Practices document (see References below)"
	Role: The element's role is missing or incorrect
	Rule ID: aria-role-missing
	....
	How To Fix
	Fix this issue by adding the appropriate, valid ARIA role or roles per the ARIA Recommendation document and the ARIA Authoring Practices document (see References below)
		
		[D.ARIA_ROLE_MISSING]: {
		shortText: "Role: The element's role is missing or incorrect",
		...
		howToFix: "Fix this issue by adding the appropriate, valid ARIA role or roles per the ARIA Recommendation document and the ARIA Authoring Practices document (see References below)"
		[W.ACCESSIBLE_ROLE]: {
		shortText: "Element does not have an appropriate role",
		issueDescText: "The element's role is missing or is not appropriate for the element's function.",
		....
		howToFix: "Fix this issue by adding the appropriate, valid ARIA role or roles per the ARIA Recommendation document and the ARIA Authoring Practices document"
			
			Understanding SC 4.1.2:
			Name, Role, Value (Level A)
			Success Criterion (SC)
			For all user interface components (including but not limited to: form elements, links and components generated by scripts), the name and role can be programmatically determined; states, properties, and values that can be set by the user can be programmatically set; and notification of changes to these items is available to user agents, including assistive technologies.
			Note
			This success criterion is primarily for Web authors who develop or script their own user interface components. For example, standard HTML controls already meet this success criterion when used according to specification.

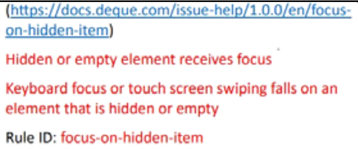
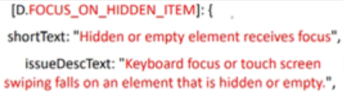
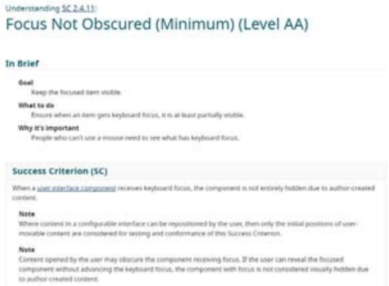
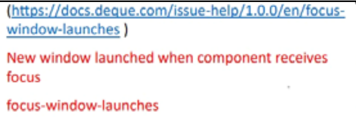
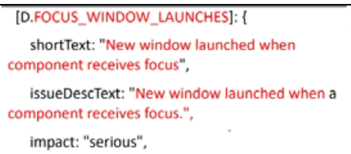
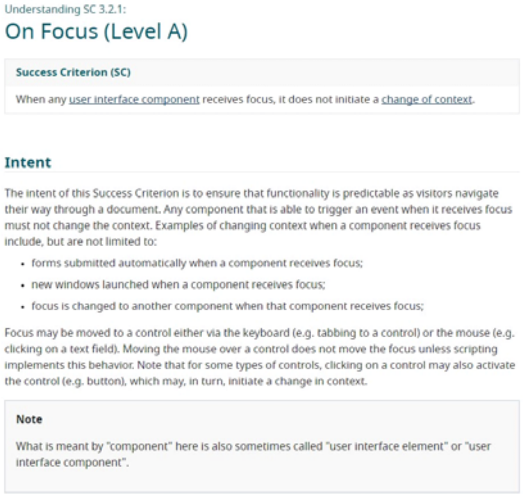
Deque Issue Help Pages // BroswerStack 1.0.0.4: "IGTRules.js"			
12	 (https://docs.deque.com/issue-help/1.0.0/en/focus-on-hidden-item) Hidden or empty element receives focus Keyboard focus or touch screen swiping falls on an element that is hidden or empty Rule ID: focus-on-hidden-item	 [D.FOCUS_ON_HIDDEN_ITEM]: { shortText: "Hidden or empty element receives focus", issueDescText: "Keyboard focus or touch screen swiping falls on an element that is hidden or empty.",	 Understanding SC 2.4.11: Focus Not Obscured (Minimum) (Level AA) In Brief Goal Keep the focused item visible. What to do Ensure when an item gets keyboard focus, it is at least partially visible. Why it's important People who can't use a mouse need to see what has keyboard focus. Success Criterion (SC) When a user interface component receives keyboard focus, the component is not entirely hidden due to author-created content. Note Where content in a configurable interface can be repositioned by the user, then only the initial positions of user-movable content are considered for testing and conformance of this Success Criterion. Note Content opened by the user may obscure the component receiving focus. If the user can reveal the focused component without advancing the keyboard focus, the component with focus is not considered visually hidden due to author-created content.
13	 (https://docs.deque.com/issue-help/1.0.0/en/focus-window-launches) New window launched when component receives focus focus-window-launches	 [D.FOCUS_WINDOW_LAUNCHES]: { shortText: "New window launched when component receives focus", issueDescText: "New window launched when a component receives focus.", impact: "serious",	 Understanding SC 3.2.1: On Focus (Level A) Success Criterion (SC) When any user interface component receives focus, it does not initiate a change of context. Intent The intent of this Success Criterion is to ensure that functionality is predictable as visitors navigate their way through a document. Any component that is able to trigger an event when it receives focus must not change the context. Examples of changing context when a component receives focus include, but are not limited to: • forms submitted automatically when a component receives focus; • new windows launched when a component receives focus; • focus is changed to another component when that component receives focus; Focus may be moved to a control either via the keyboard (e.g. tabbing to a control) or the mouse (e.g. clicking on a text field). Moving the mouse over a control does not move the focus unless scripting implements this behavior. Note that for some types of controls, clicking on a control may also activate the control (e.g. button), which may, in turn, initiate a change in context. Note What is meant by "component" here is also sometimes called "user interface element" or "user interface component".

Exhibit 1: Comparison of Asserted Deque Help Code, Accused BrowserStack Help Text Code, and WCAG Standards

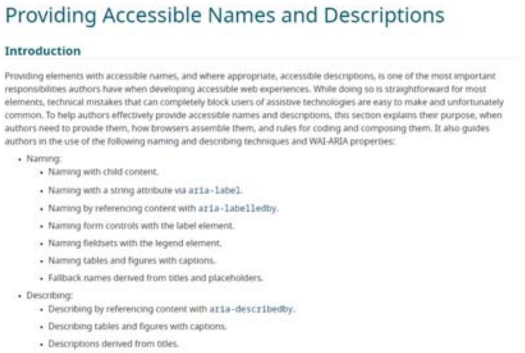
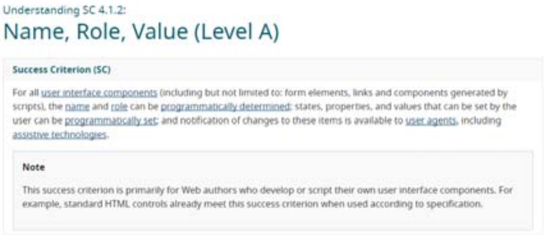
Code Segment	Asserted Deque Help Text Code	Accused BrowserStack Help Text Code	WCAG Standard
14	(https://docs.deque.com/issue-help/1.0.0/en/aria-name-missing-incorrect) The element's name is missing or incorrect The element's accessible name is missing or is not appropriate for the element's purpose Rule ID: aria-name-missing-incorrect	[W.ACCESSIBLE_NAME]: { shortText: "Element does not have an appropriate accessible name", issueDescText: "The element's accessible name is missing or is not appropriate for the element's purpose.",	 Providing Accessible Names and Descriptions **Introduction** Providing elements with accessible names, and where appropriate, accessible descriptions, is one of the most important responsibilities authors have when developing accessible web experiences. While doing so is straightforward for most elements, technical mistakes that can completely block users of assistive technologies are easy to make and unfortunately common. To help authors effectively provide accessible names and descriptions, this section explains their purpose, when authors need to provide them, how browsers assemble them, and rules for coding and composing them. It also guides authors in the use of the following naming and describing techniques and WAI-ARIA properties: • Naming: • Naming with child content. • Naming with a string attribute via aria-label. • Naming by referencing content with aria-labelledby. • Naming form controls with the label element. • Naming fieldsets with the legend element. • Naming tables and figures with captions. • Fallback names derived from titles and placeholders. • Describing: • Describing by referencing content with aria-describedby. • Describing tables and figures with captions. • Descriptions derived from titles.
15	(https://docs.deque.com/issue-help/1.0.0/en/aria-state-property-missing) The element has missing or incorrect states or properties The element has missing or incorrect states or properties that are necessary for screen reader users to interact with or understand the content conveyed by the element How to Fix ... Determine the role of the component: Identify the role of the component by looking at the HTML code and checking whether it has the appropriate role attribute. The role attribute defines the type of the component and its expected behavior. ... Apply the ARIA attributes: Apply the appropriate ARIA attributes to the component using HTML. For example, if a "textbox" component has an error state, you can add the aria-invalid attribute with a value of "true". ...	[W.ACCESSIBLE_STATES]: { shortText: "Element has missing or incorrect states", issueDescText: "The element has missing or incorrect states or properties that are necessary for screen reader users to interact with or understand the content conveyed by the element.", ... howToFix: "Fix this issue by determining the role of the component and applying the appropriate ARIA attributes to the component using HTML"	 Understanding SC 4.1.2: Name, Role, Value (Level A) **Success Criterion (SC)** For all user interface components (including but not limited to: form elements, links and components generated by scripts), the name and role can be programmatically determined; states, properties, and values that can be set by the user can be programmatically set; and notification of changes to these items is available to user agents, including assistive technologies. **Note** This success criterion is primarily for Web authors who develop or script their own user interface components. For example, standard HTML controls already meet this success criterion when used according to specification.

Exhibit 1 Notes

1. The W3C requires that websites be “Keyboard Accessible,” and the title used to describe the failure mode, “keyboard inaccessible,” is straightforwardly derived from it.[1] The relevant UAAG standard also requires that “any … functionality” must be “available” “through keyboard input alone,” just like the accused Help Text short description.[2] Finally, the relevant WCAG Technique explains that an author should verify that “each of the functions identified can be performed using only the keyboard,” just like the accused description text, “There is no way to perform the function using only the keyboard.”[3]

2. The relevant WCAG standard requires that “the keyboard focus indicator is visible,”[4] and a related technique notes that the standard is failed when the “visual focus indicator” “is turned off or rendered non-visible.”[5] The only difference between the relevant text and the terms from the standard is the replacement of “turned off or rendered non-visible” with the shorter “missing altogether.”

3. The W3C notes that developers should ensure that “[w]hen a user interface component receives keyboard focus, the component is not entirely hidden due to author-created content.”[6] Again, the help text is written in the negative, explaining that there is an issue when a hidden or empty element receives the focus.

4. The W3C instructs that a developer must “[g]ive components correct … [ARIA] roles.”[7] W3C standards specify a list of appropriate roles for various HTML elements,[8] and require that “the characteristics of the user interface component are described by the role,” i.e., that they are appropriate for the relevant element, as the help text explains.[9]

5. W3C techniques suggest that a developer “use WAI-ARIA state and property attributes to expose the state, properties and values of a user interface component….” to meet the standards.[10] Developers must set these values because “the state and property attributes enables users to understand the widget and how to interact with it,”[11] just as the help text explains.

[1] https://www.w3.org/WAI/WCAG21/Understanding/keyboard-accessible

[2] https://www.w3.org/TR/2002/REC-UAAG10-20021217/guidelines#tech-device-independent-ui

[3] https://www.w3.org/TR/2010/NOTE-WCAG20-TECHS-20101014/G202

[4] https://www.w3.org/WAI/WCAG21/Understanding/focus-visible

[5] https://www.w3.org/WAI/WCAG21/Techniques/failures/F78

[6] https://www.w3.org/TR/WCAG22/#focus-not-obscured-minimum; *see also* https://www.w3.org/WAI/WCAG21/Techniques/failures/F110 (failure on “hiding focused elements”)

[7] https://www.w3.org/WAI/WCAG22/Understanding/name-role-value.html

[8] https://www.w3.org/TR/html-aria/#docconformance

[9] https://www.w3.org/WAI/WCAG21/Techniques/aria/ARIA4.html

[10] https://www.w3.org/WAI/WCAG21/Techniques/aria/ARIA5.html

[11] https://www.w3.org/WAI/WCAG22/Techniques/aria/ARIA5.html; *see also* https://www.w3.org/TR/wai-aria-1.2/#requiredState

6. Here, the WCAG explains the "on focus" criterion with the nearly-identical as in the help text: the standard is not met when "new windows launched when a component receives focus."[12]

7. The W3C instructs that an "accessible name" should "convey the purpose or intent of the element."[13] The help text simply restates that each element should have an accessible name and that it should appropriately convey the purpose.

8. This Help Text is not related to a standard, but is basic user interface text that asks a user to state whether some functionality not reachable through the keyboard has an alternative or not and is properly within the tab order. That is, even though this is not related to a standard as in other portions of the Help Text, this text is reflects a functional description of what the tool is testing.

9. As described for example #1 above, the text outside of the "howToFix" portion is directly derived from the standards. With respect to the howToFix text, the W3C suggests that the standard can be met through techniques that include "Using HTML form controls and links," as in the help text. A related technique document describes the reasoning with nearly identical language to that in the help text:

   > User agents provide the keyboard operation of HTML form controls and links. In addition, the user agent maps the form controls and links to an accessibility API. Assistive technologies use the accessibility API to extract appropriate accessibility information, such as role, name, state, and value, and present them to users.

   The technique document goes on to note various "HTML links and form controls" including, in order: "<a>", "<button>," "<fieldset>", "<input>" (several types including those listed in the help text), and "<select>". I note that, with the exception of "fieldset," this is the same set of elements described in the help text and in the same order.[14]

10. The solutions presented for fixing the error here are found in the WCAG guidelines and suggestions. The W3C suggests that "[w]hen a user interface control receives focus, a visible border is displayed around it."[15] It similarly suggests that a developer "apply a background color when the link elements receive focus;" the same reference also suggests a "3:1 contrast ratio," as does the help text.[16] Another technique document suggests a "2px" (two pixel) box,[17]and another suggests that a 1px border is the minimum or "at least 4 CSS pixels along the shortest side of the component" (a thick line

[12] https://www.w3.org/WAI/WCAG21/Understanding/on-focus.html
[13] https://www.w3.org/WAI/ARIA/apg/practices/names-and-descriptions/
[14] https://www.w3.org/WAI/WCAG21/Techniques/html/H91
[15] https://www.w3.org/WAI/WCAG21/Understanding/focus-visible
[16] https://www.w3.org/WAI/WCAG21/Techniques/css/C15;
https://www.w3.org/WAI/WCAG21/Understanding/contrast-minimum.html
[17] https://www.w3.org/WAI/WCAG21/Techniques/css/C40

next to the element).[18] Yet another promotes "[u]sing the default focus indicator for the platform," as does the help text.[19]

11. *See* #4 above. The help text here does little more than tell the developer to refer to the relevant documentation for ARIA roles, such as those I identified above with respect to excerpt #4.

12. The text in this excerpt is redundant of the text above in excerpt #3.

13. The text in this excerpt is redundant of the text above in excerpt #6

14. The text in this excerpt is redundant of the text above in excerpt #7

15. The text in this excerpt is largely redundant of the text above in excerpt #5. The additional information on how to fix the problem simply instructs the developer to add correct roles and ARIA attributes, as the documents linked above instruct.

[18] https://www.w3.org/WAI/WCAG21/Techniques/general/G195 ("focus indicator area is at least the size of a 1 CSS px border around the component")

[19] https://www.w3.org/WAI/WCAG22/Techniques/general/G165